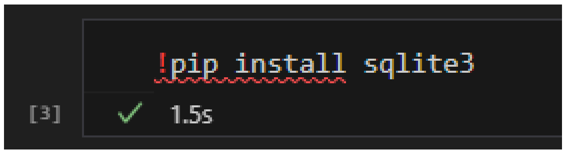
Task 1:

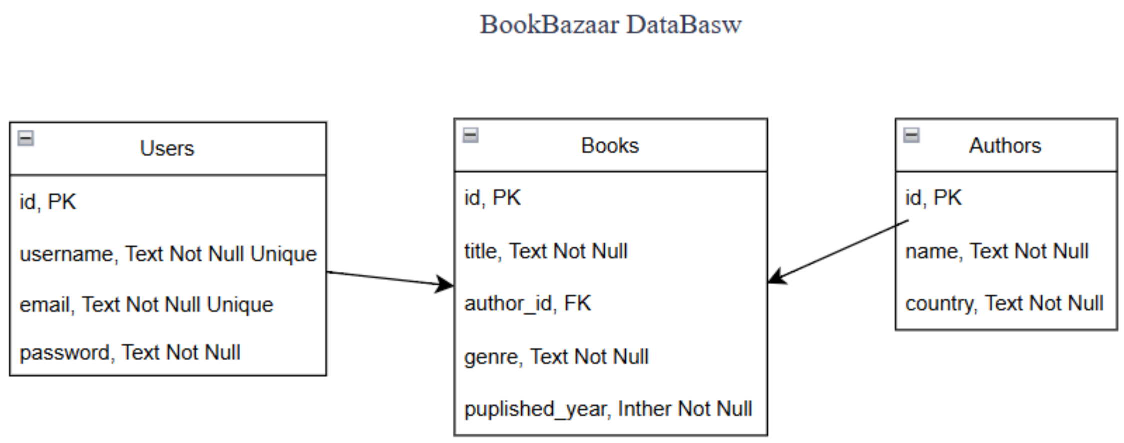


```
!pip install sqlite3
```

Task 2:

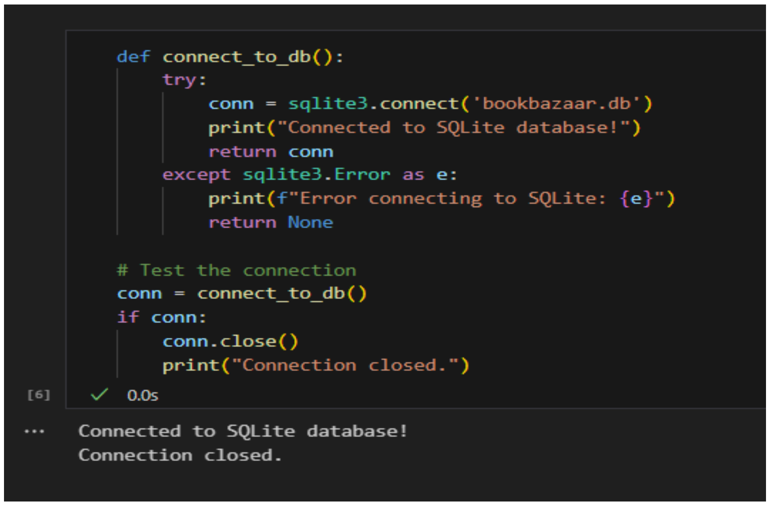
Design the Relational Database Schema

Create an ER diagram to visualize your schema



Task 3:

Connect Python to SQLite



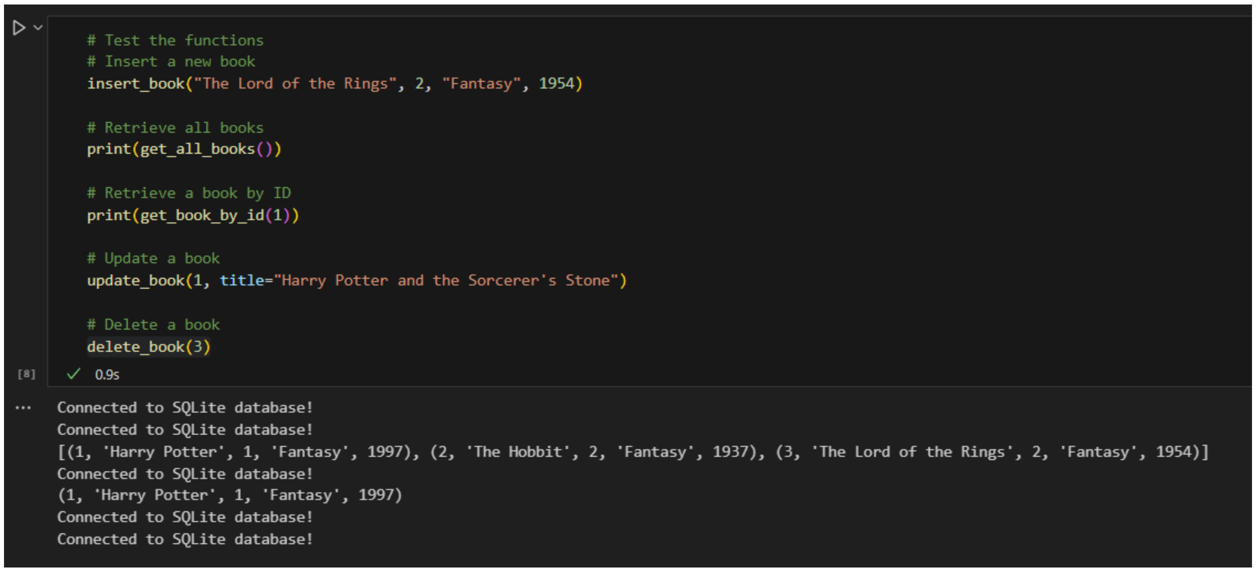
```
def connect_to_db():
    try:
        conn = sqlite3.connect('bookbazaar.db')
        print("Connected to SQLite database!")
        return conn
    except sqlite3.Error as e:
        print(f"Error connecting to SQLite: {e}")
        return None

# Test the connection
conn = connect_to_db()
if conn:
    conn.close()
    print("Connection closed.")
```

```
Connected to SQLite database!
Connection closed.
```

Task 4:

Title: Implement CRUD Operations on SQLite via Python



```
# Test the functions
# Insert a new book
insert_book("The Lord of the Rings", 2, "Fantasy", 1954)

# Retrieve all books
print(get_all_books())

# Retrieve a book by ID
print(get_book_by_id(1))

# Update a book
update_book(1, title="Harry Potter and the Sorcerer's Stone")

# Delete a book
delete_book(3)
```

[8] ✓ 0.9s

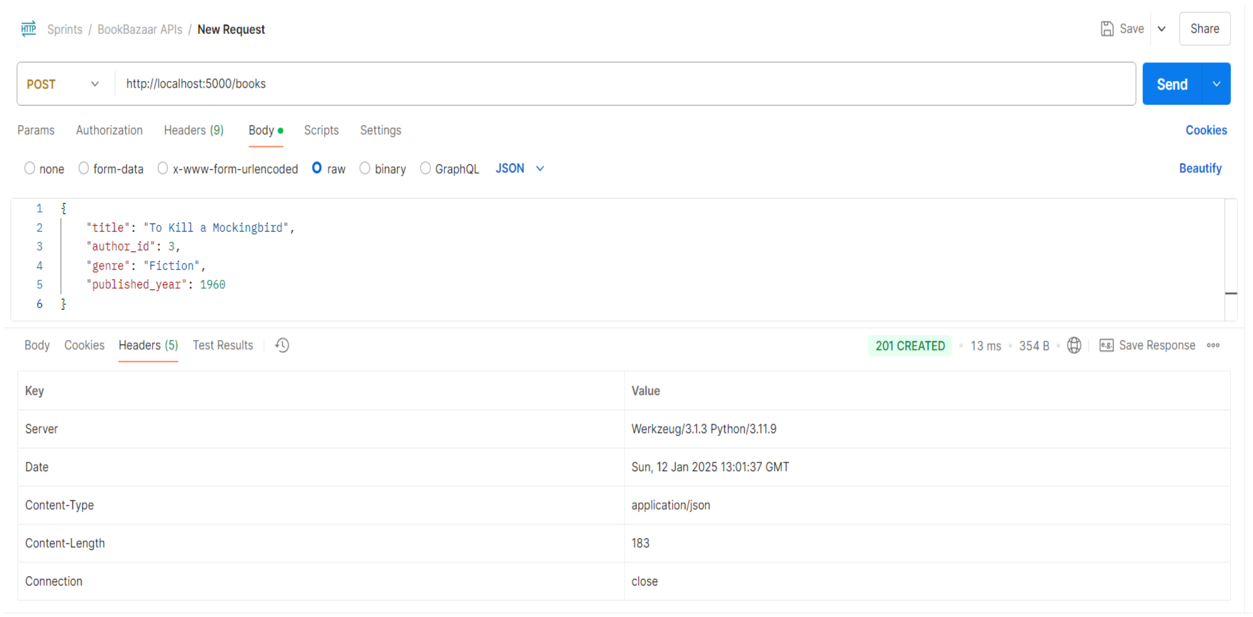
```
Connected to SQLite database!
Connected to SQLite database!
[(1, 'Harry Potter', 1, 'Fantasy', 1997), (2, 'The Hobbit', 2, 'Fantasy', 1937), (3, 'The Lord of the Rings', 2, 'Fantasy', 1954)]
Connected to SQLite database!
(1, 'Harry Potter', 1, 'Fantasy', 1997)
Connected to SQLite database!
Connected to SQLite database!
```

Task 5:

Develop RESTful APIs with Python

Task 6: Test APIs Using Postman

1. Add a New Book (POST /books)



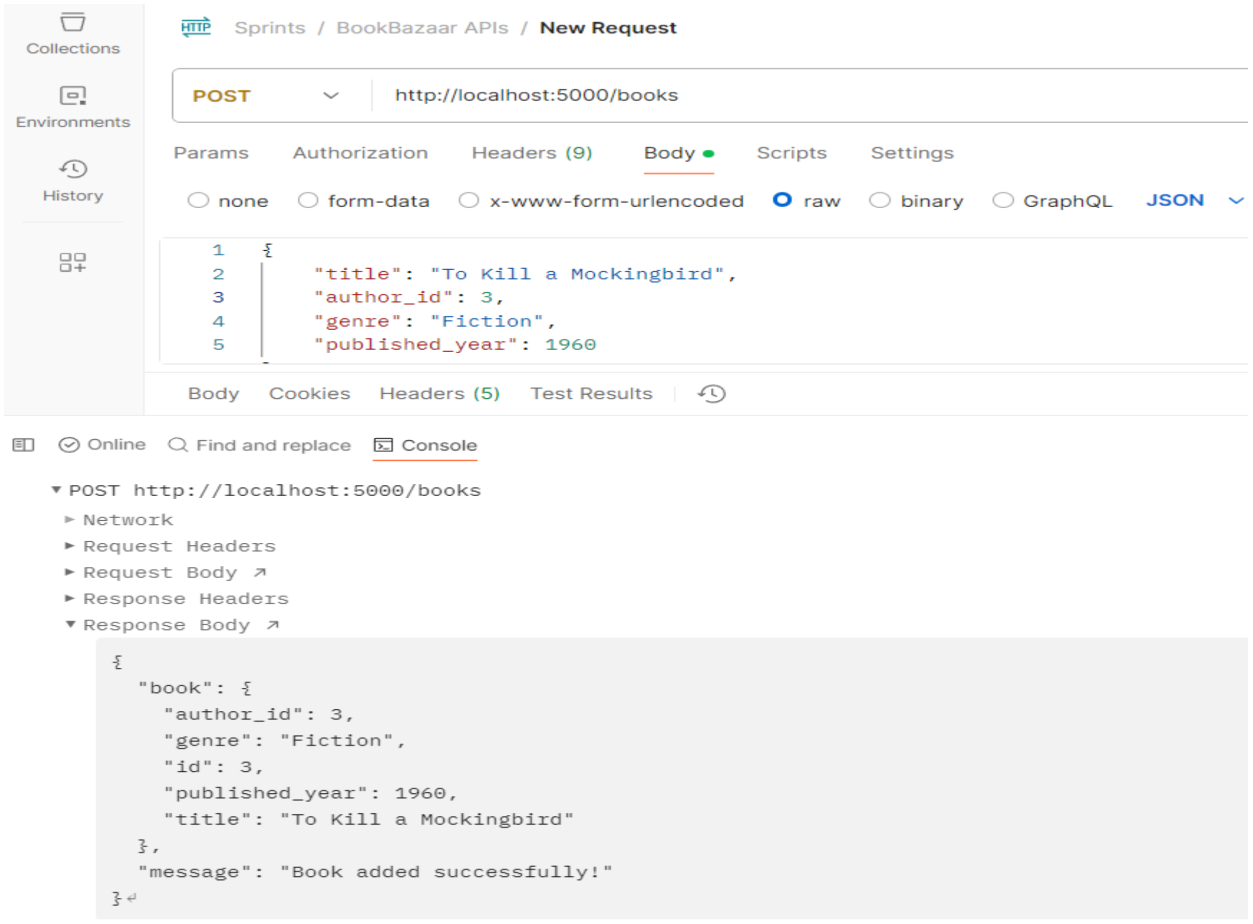
POST http://localhost:5000/books

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
    "title": "To Kill a Mockingbird",
    "author_id": 3,
    "genre": "Fiction",
    "published_year": 1960
}
```

201 CREATED • 13 ms • 354 B

Key	Value
Server	Werkzeug/3.1.3 Python/3.11.9
Date	Sun, 12 Jan 2025 13:01:37 GMT
Content-Type	application/json
Content-Length	183
Connection	close



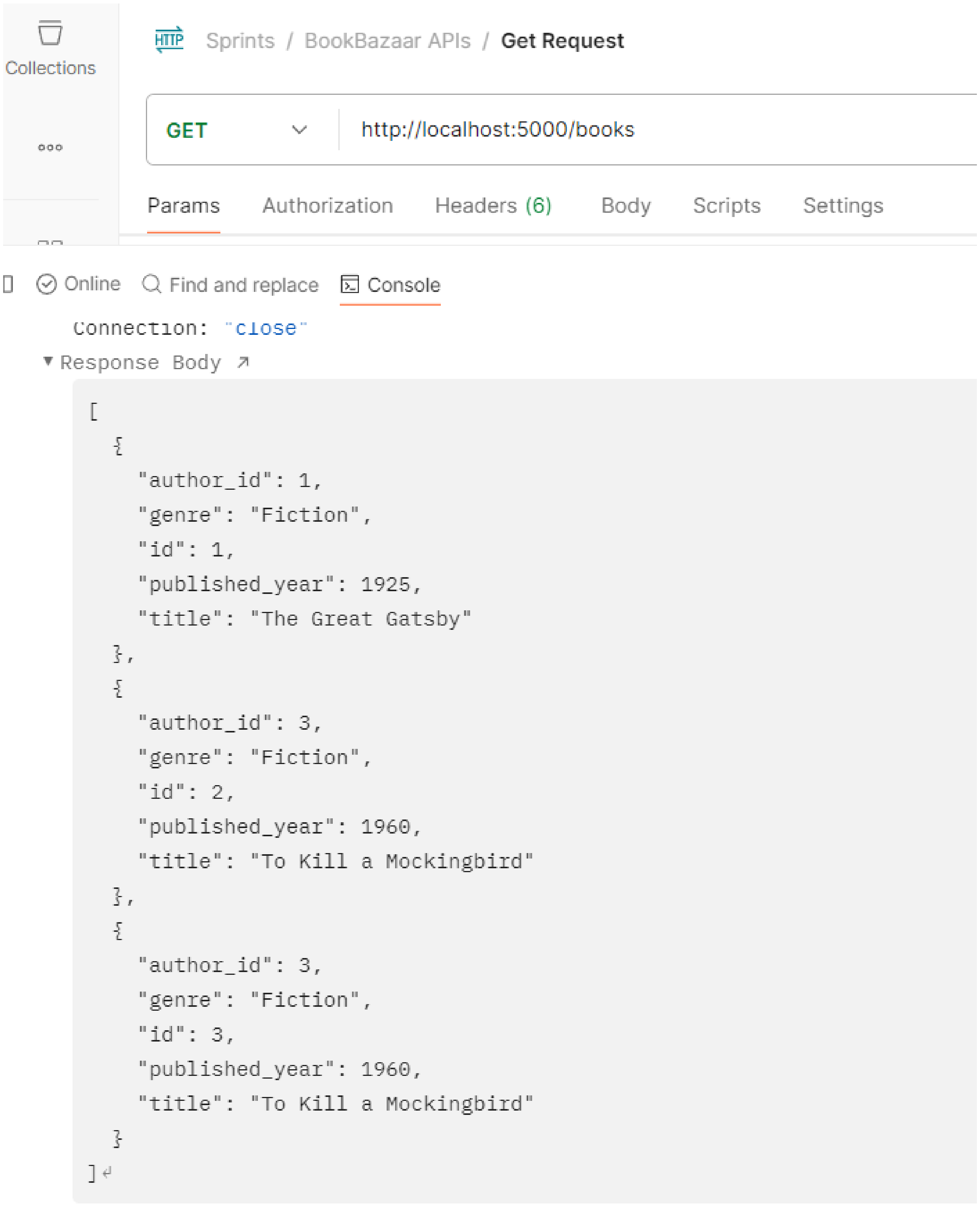
POST http://localhost:5000/books

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
    "title": "To Kill a Mockingbird",
    "author_id": 3,
    "genre": "Fiction",
    "published_year": 1960
```

```
▼ POST http://localhost:5000/books
  ► Network
  ► Request Headers
  ► Request Body ↗
  ► Response Headers
  ▼ Response Body ↗
    {
      "book": {
        "author_id": 3,
        "genre": "Fiction",
        "id": 3,
        "published_year": 1960,
        "title": "To Kill a Mockingbird"
      },
      "message": "Book added successfully!"
    }
```

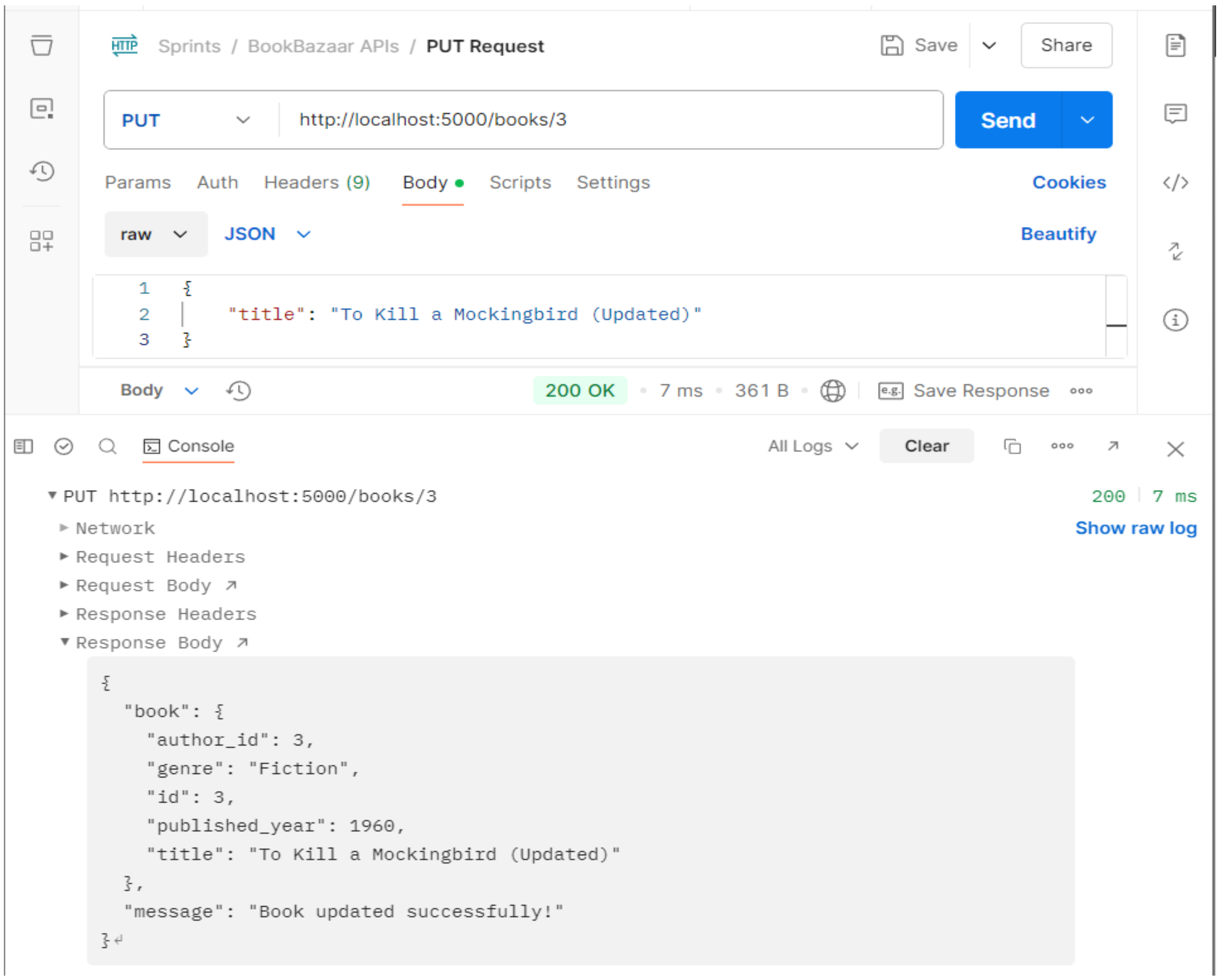
2. Retrieve the List of Books (GET /books)



GET http://localhost:5000/books

```
  Connection: "close"
▼ Response Body ↗
[
  {
    "author_id": 1,
    "genre": "Fiction",
    "id": 1,
    "published_year": 1925,
    "title": "The Great Gatsby"
  },
  {
    "author_id": 3,
    "genre": "Fiction",
    "id": 2,
    "published_year": 1960,
    "title": "To Kill a Mockingbird"
  },
  {
    "author_id": 3,
    "genre": "Fiction",
    "id": 3,
    "published_year": 1960,
    "title": "To Kill a Mockingbird"
  }
]↵
```

3. Update a Book's Details (PUT /books/<id>)



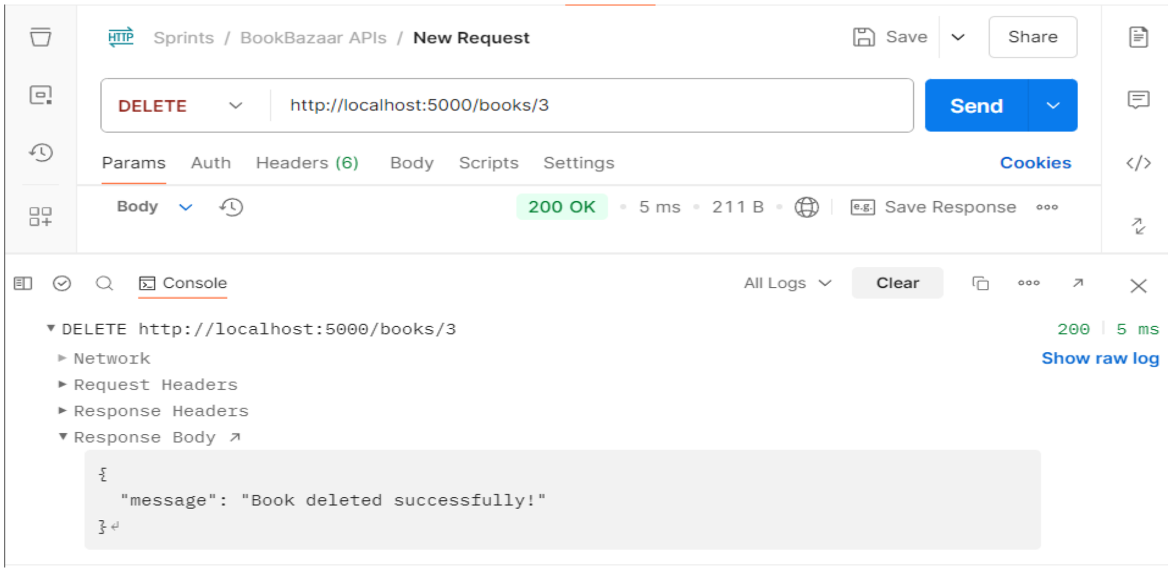
PUT http://localhost:5000/books/3

```
{
  "title": "To Kill a Mockingbird (Updated)"
}
```

200 OK • 7 ms • 361 B

```
▼ PUT http://localhost:5000/books/3                                   200 | 7 ms
  ▸ Network
  ▸ Request Headers
  ▸ Request Body ↗
  ▸ Response Headers
  ▾ Response Body ↗
    {
      "book": {
        "author_id": 3,
        "genre": "Fiction",
        "id": 3,
        "published_year": 1960,
        "title": "To Kill a Mockingbird (Updated)"
      },
      "message": "Book updated successfully!"
    }
```

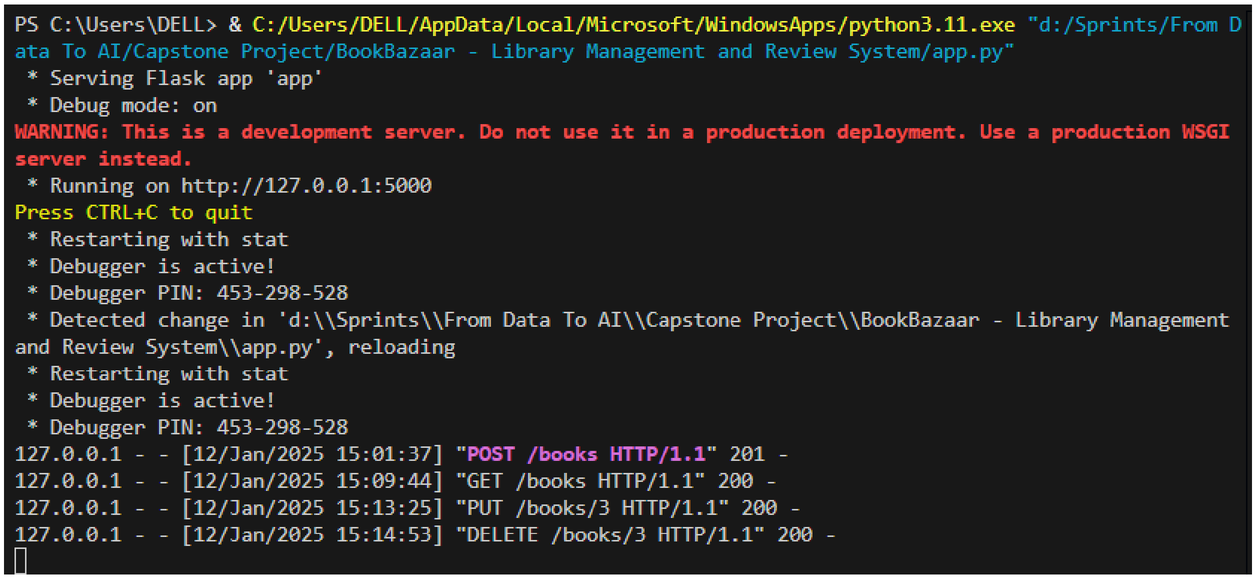
4. Delete a Book (DELETE /books/<id>)



DELETE http://localhost:5000/books/3

200 OK • 5 ms • 211 B

```
▼ DELETE http://localhost:5000/books/3                                200 | 5 ms
  ▸ Network
  ▸ Request Headers
  ▸ Response Headers
  ▾ Response Body ↗
    {
      "message": "Book deleted successfully!"
    }
```

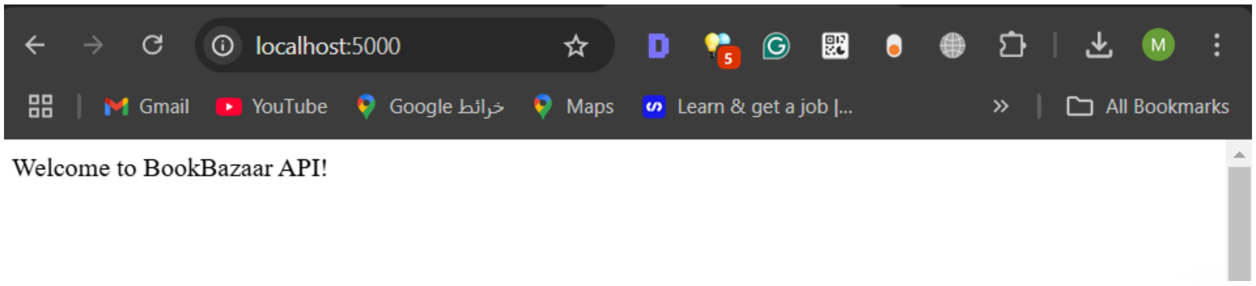


```
PS C:\Users\DELL> & C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/Sprints/From D
ata To AI/Capstone Project/BookBazaar - Library Management and Review System/app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 453-298-528
 * Detected change in 'd:\\Sprints\\From Data To AI\\Capstone Project\\BookBazaar - Library Management
and Review System\\app.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 453-298-528
127.0.0.1 - - [12/Jan/2025 15:01:37] "POST /books HTTP/1.1" 201 -
127.0.0.1 - - [12/Jan/2025 15:09:44] "GET /books HTTP/1.1" 200 -
127.0.0.1 - - [12/Jan/2025 15:13:25] "PUT /books/3 HTTP/1.1" 200 -
127.0.0.1 - - [12/Jan/2025 15:14:53] "DELETE /books/3 HTTP/1.1" 200 -
```

Task 7:

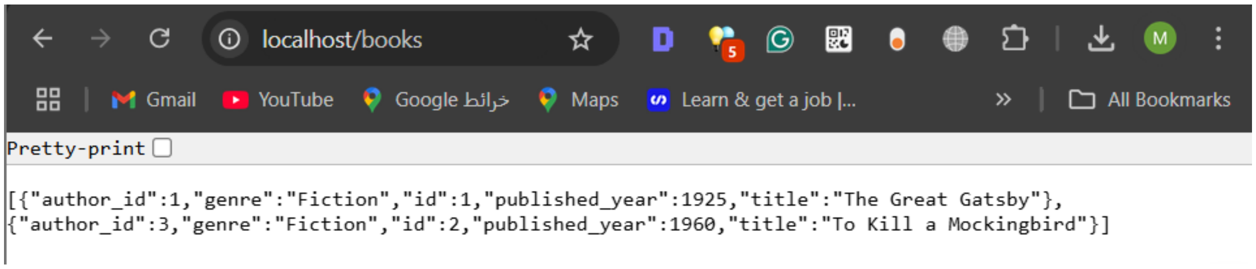
Access Your API:

Host APIs on Apache Web Server : http://localhost:5000



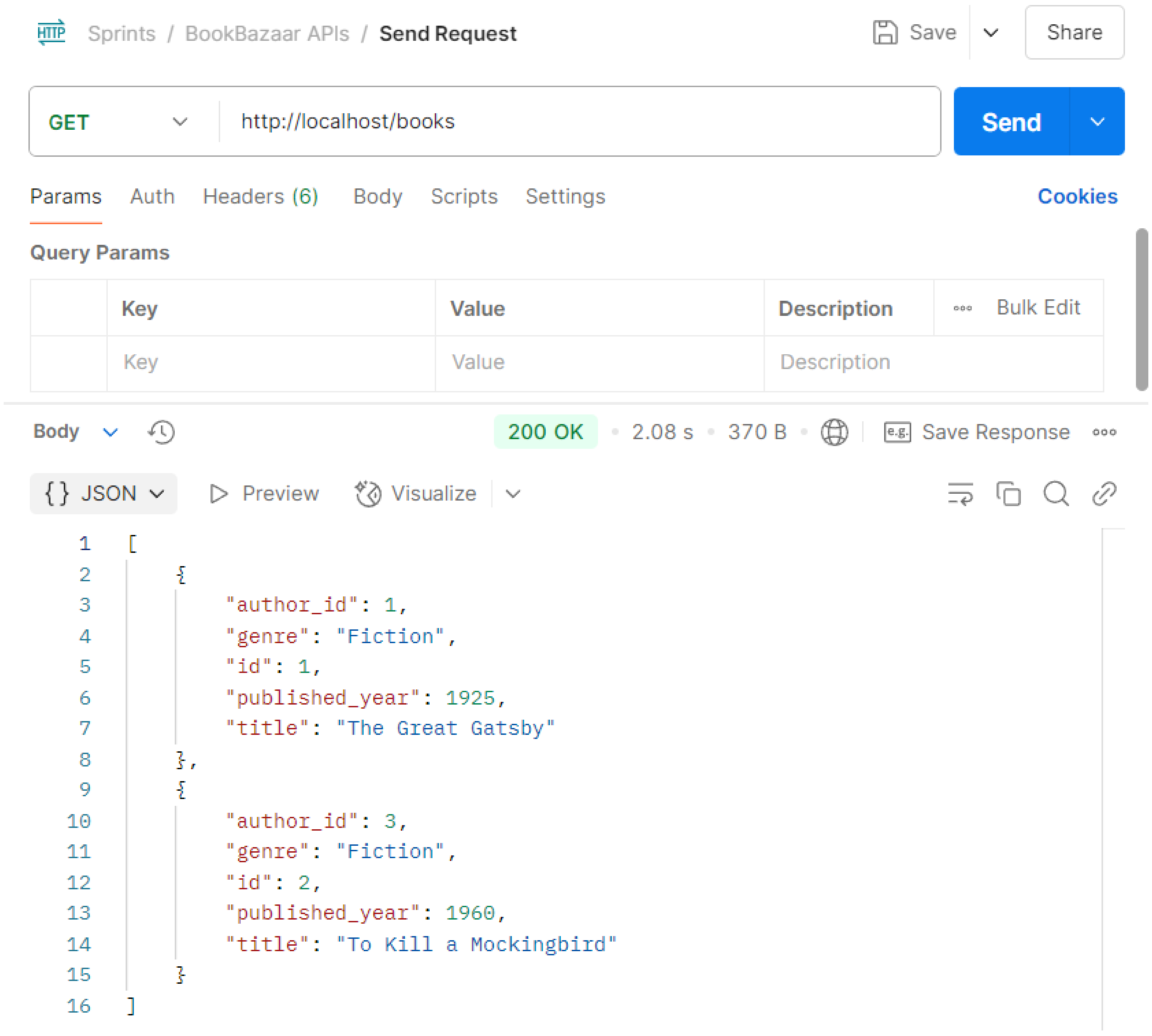
Welcome to BookBazaar API!

Ensure that when you access your server’s URL in a browser : http://localhost/books



```
[{"author_id":1,"genre":"Fiction","id":1,"published_year":1925,"title":"The Great Gatsby"},
{"author_id":3,"genre":"Fiction","id":2,"published_year":1960,"title":"To Kill a Mockingbird"}]
```

Test with Postman



GET http://localhost/books

Query Params

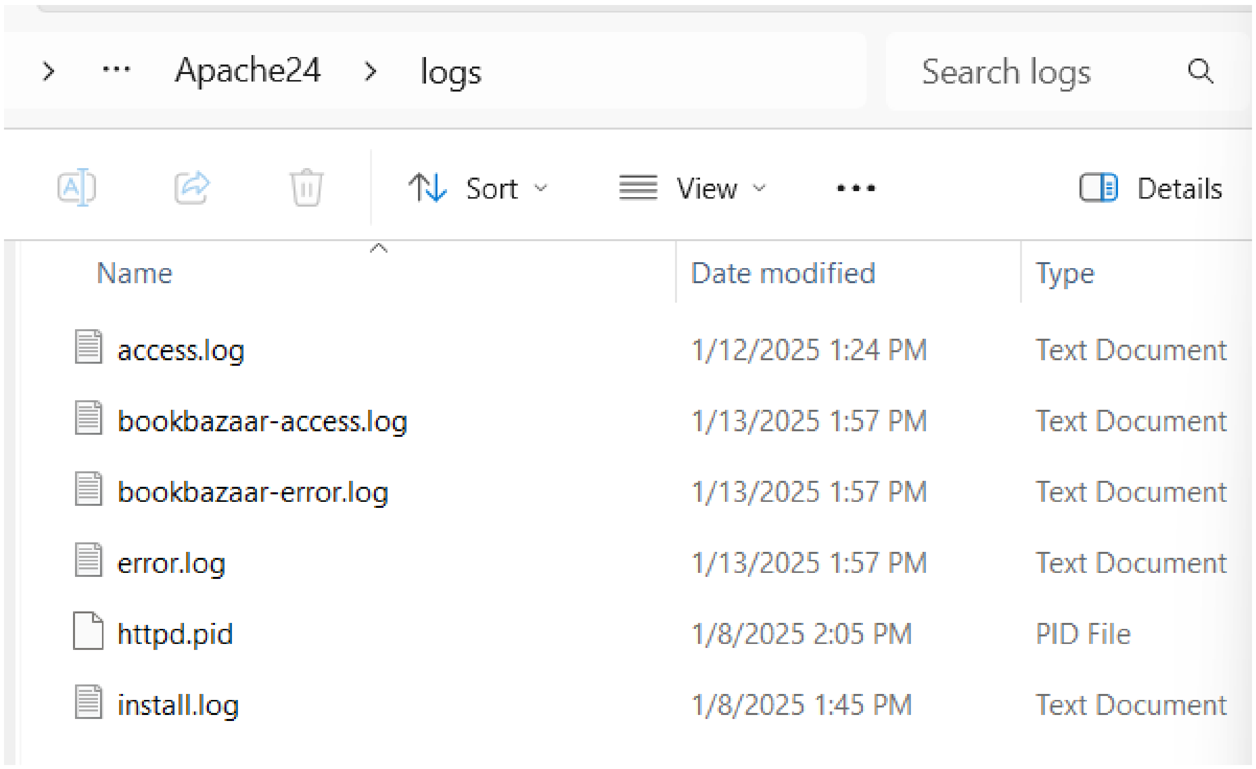
Key	Value	Description
Key	Value	Description

200 OK • 2.08 s • 370 B

```
[
    {
        "author_id": 1,
        "genre": "Fiction",
        "id": 1,
        "published_year": 1925,
        "title": "The Great Gatsby"
    },
    {
        "author_id": 3,
        "genre": "Fiction",
        "id": 2,
        "published_year": 1960,
        "title": "To Kill a Mockingbird"
    }
]
```

Troubleshoot any server configuration issues (e.g., permission issues, missing modules).

- Error log: "C:\Apache24\logs\bookbazaar-error.log"
- Access log: "C:\Apache24\logs\bookbazaar-access.log"

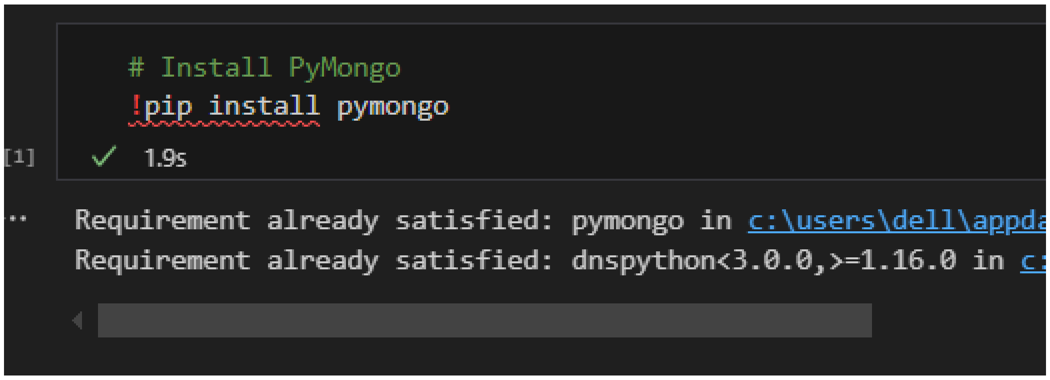


Name	Date modified	Type
access.log	1/12/2025 1:24 PM	Text Document
bookbazaar-access.log	1/13/2025 1:57 PM	Text Document
bookbazaar-error.log	1/13/2025 1:57 PM	Text Document
error.log	1/13/2025 1:57 PM	Text Document
httpd.pid	1/8/2025 2:05 PM	PID File
install.log	1/8/2025 1:45 PM	Text Document

Task 8, 9:

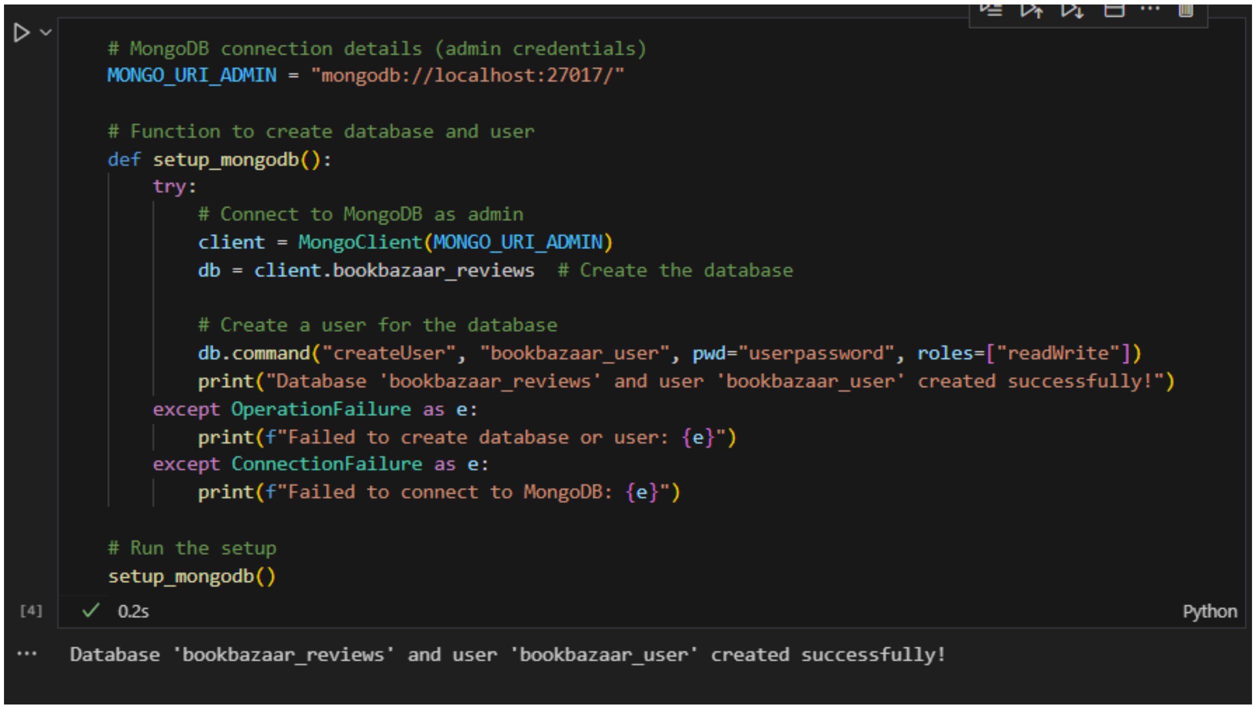
Title: Set Up the Non-Relational Database with MongoDB

Connect Python to MongoDB Using PyMongo



```
# Install PyMongo
!pip install pymongo
[1] ✓ 1.9s

Requirement already satisfied: pymongo in c:\users\dell\appda
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in c:
```



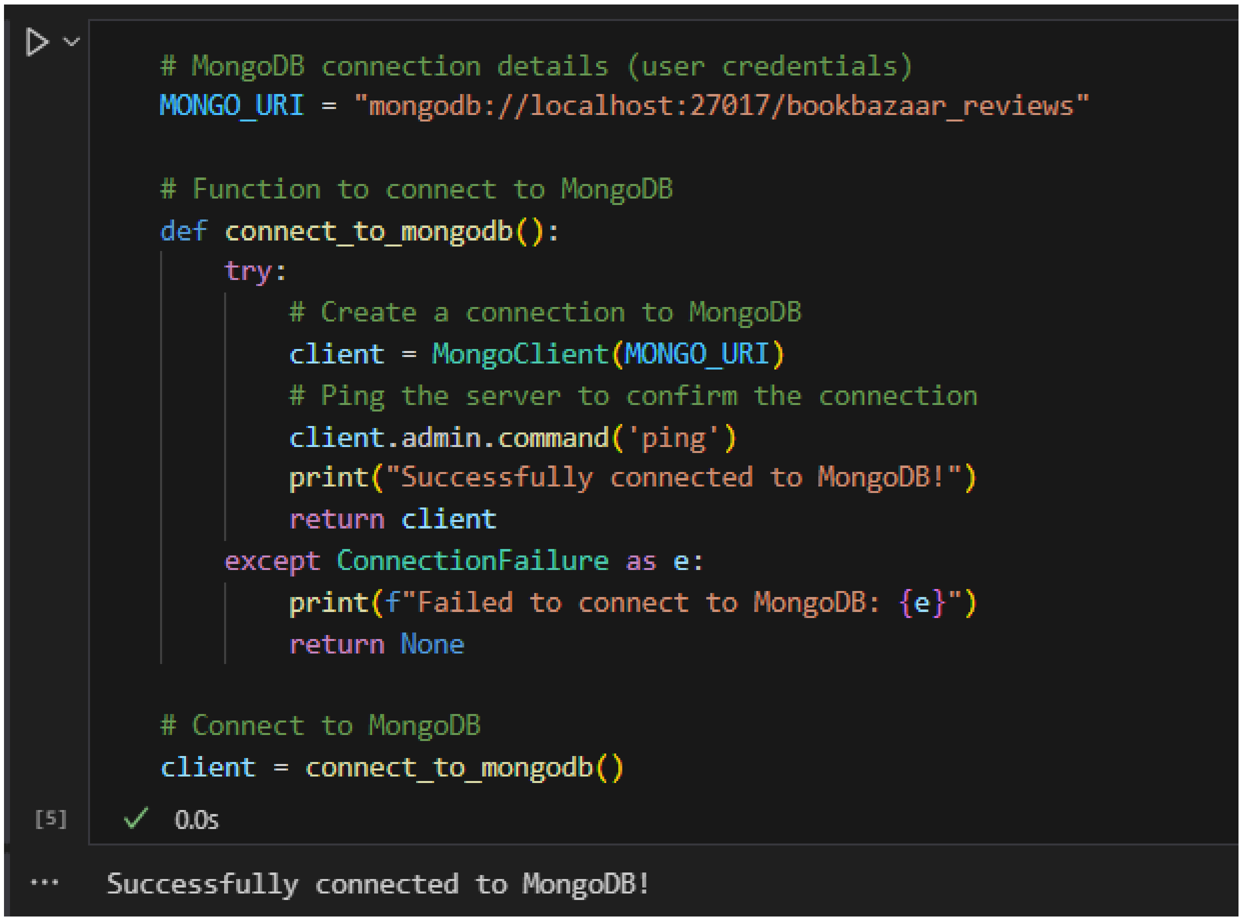
```
# MongoDB connection details (admin credentials)
MONGO_URI_ADMIN = "mongodb://localhost:27017/"

# Function to create database and user
def setup_mongodb():
    try:
        # Connect to MongoDB as admin
        client = MongoClient(MONGO_URI_ADMIN)
        db = client.bookbazaar_reviews  # Create the database

        # Create a user for the database
        db.command("createUser", "bookbazaar_user", pwd="userpassword", roles=["readWrite"])
        print("Database 'bookbazaar_reviews' and user 'bookbazaar_user' created successfully!")
    except OperationFailure as e:
        print(f"Failed to create database or user: {e}")
    except ConnectionFailure as e:
        print(f"Failed to connect to MongoDB: {e}")

# Run the setup
setup_mongodb()
[4] ✓ 0.2s                                                          Python

Database 'bookbazaar_reviews' and user 'bookbazaar_user' created successfully!
```



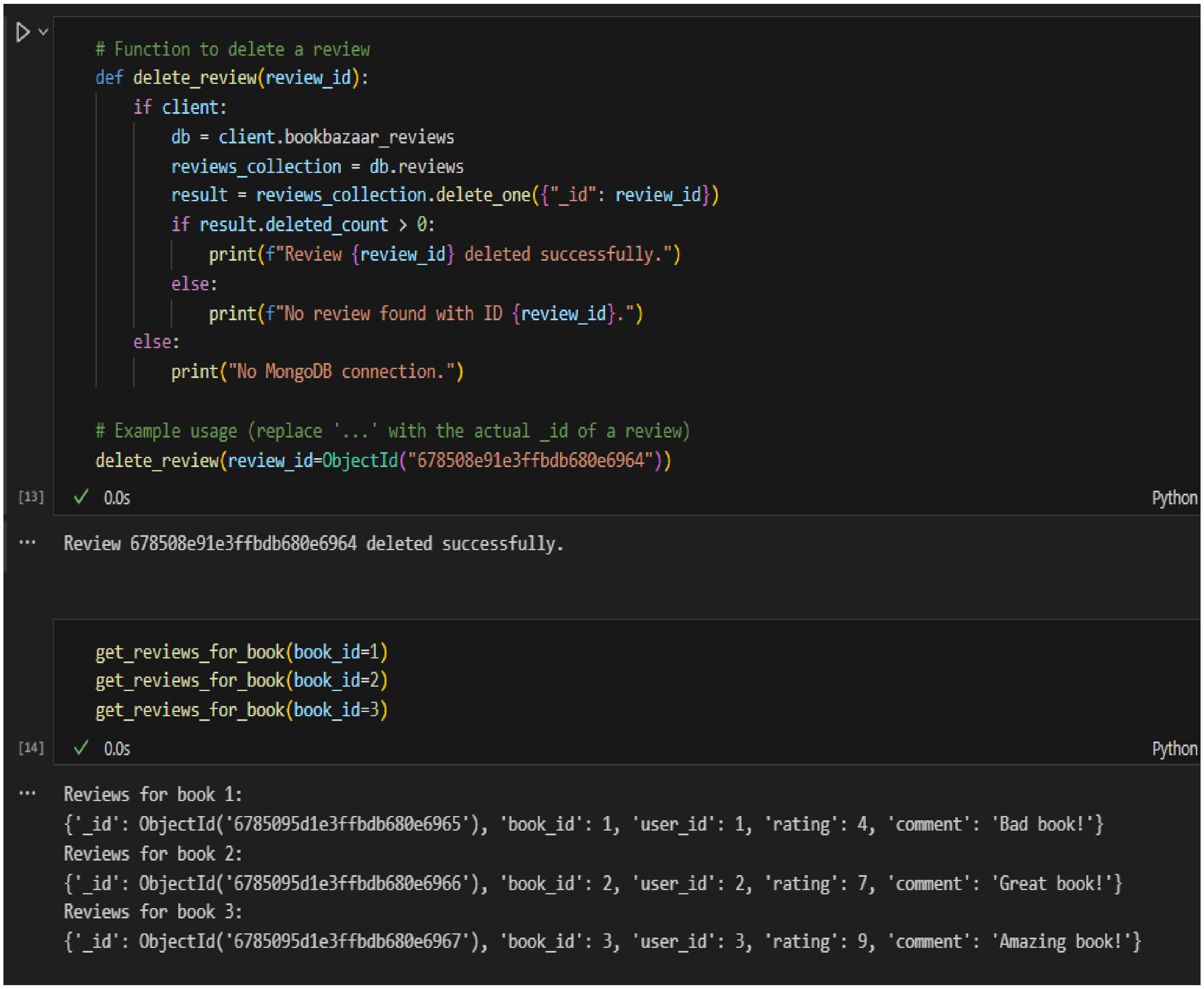
```python
# MongoDB connection details (user credentials)
MONGO_URI = "mongodb://localhost:27017/bookbazaar_reviews"

# Function to connect to MongoDB
def connect_to_mongodb():
    try:
        # Create a connection to MongoDB
        client = MongoClient(MONGO_URI)
        # Ping the server to confirm the connection
        client.admin.command('ping')
        print("Successfully connected to MongoDB!")
        return client
    except ConnectionFailure as e:
        print(f"Failed to connect to MongoDB: {e}")
        return None

# Connect to MongoDB
client = connect_to_mongodb()
```

[5] ✓ 0.0s

```
Successfully connected to MongoDB!
```



```python
# Function to delete a review
def delete_review(review_id):
    if client:
        db = client.bookbazaar_reviews
        reviews_collection = db.reviews
        result = reviews_collection.delete_one({"_id": review_id})
        if result.deleted_count > 0:
            print(f"Review {review_id} deleted successfully.")
        else:
            print(f"No review found with ID {review_id}.")
    else:
        print("No MongoDB connection.")

# Example usage (replace '...' with the actual _id of a review)
delete_review(review_id=ObjectId("678508e91e3ffbdb680e6964"))
```

[13] ✓ 0.0s Python

```
Review 678508e91e3ffbdb680e6964 deleted successfully.
```

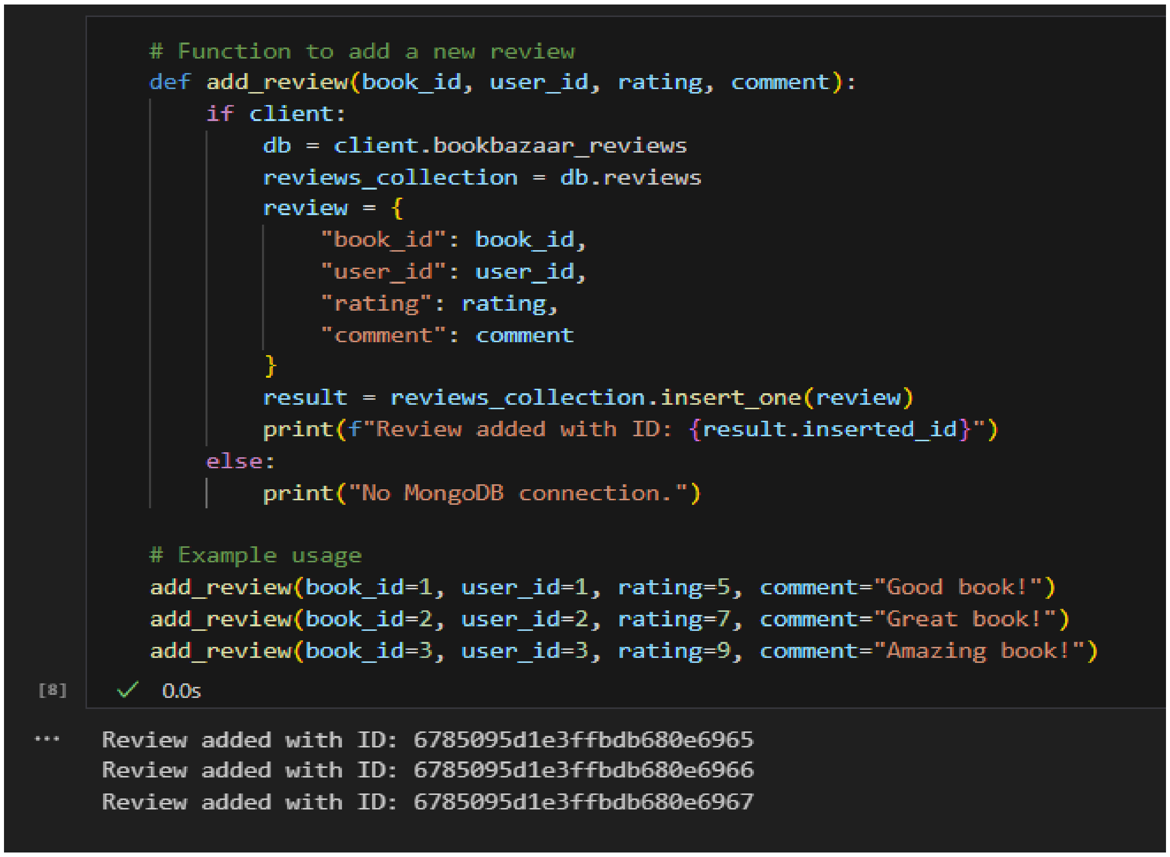
```python
get_reviews_for_book(book_id=1)
get_reviews_for_book(book_id=2)
get_reviews_for_book(book_id=3)
```

[14] ✓ 0.0s Python

```
Reviews for book 1:
{'_id': ObjectId('6785095d1e3ffbdb680e6965'), 'book_id': 1, 'user_id': 1, 'rating': 4, 'comment': 'Bad book!'}
Reviews for book 2:
{'_id': ObjectId('6785095d1e3ffbdb680e6966'), 'book_id': 2, 'user_id': 2, 'rating': 7, 'comment': 'Great book!'}
Reviews for book 3:
{'_id': ObjectId('6785095d1e3ffbdb680e6967'), 'book_id': 3, 'user_id': 3, 'rating': 9, 'comment': 'Amazing book!'}
```

Task 10:

Implement CRUD Operations on MongoDB via Python

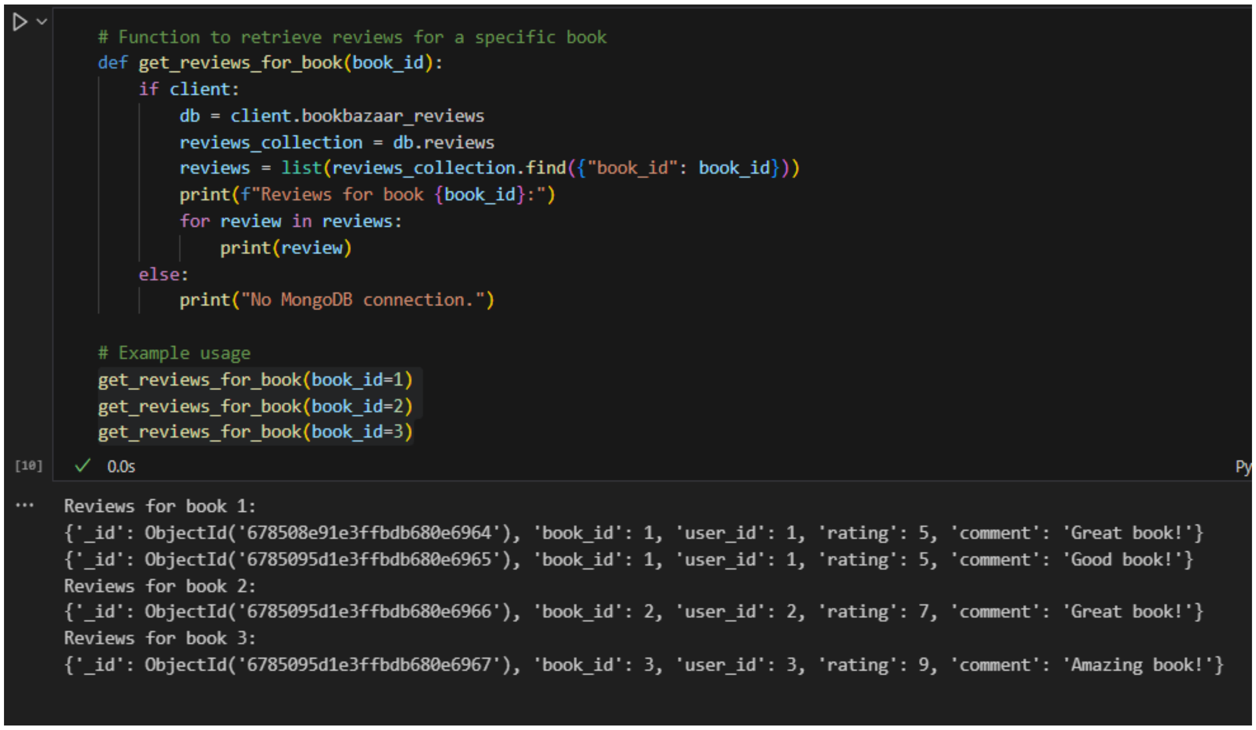


```
# Function to add a new review
def add_review(book_id, user_id, rating, comment):
    if client:
        db = client.bookbazaar_reviews
        reviews_collection = db.reviews
        review = {
            "book_id": book_id,
            "user_id": user_id,
            "rating": rating,
            "comment": comment
        }
        result = reviews_collection.insert_one(review)
        print(f"Review added with ID: {result.inserted_id}")
    else:
        print("No MongoDB connection.")

# Example usage
add_review(book_id=1, user_id=1, rating=5, comment="Good book!")
add_review(book_id=2, user_id=2, rating=7, comment="Great book!")
add_review(book_id=3, user_id=3, rating=9, comment="Amazing book!")
```

[8] ✓ 0.0s

```
Review added with ID: 6785095d1e3ffbdb680e6965
Review added with ID: 6785095d1e3ffbdb680e6966
Review added with ID: 6785095d1e3ffbdb680e6967
```

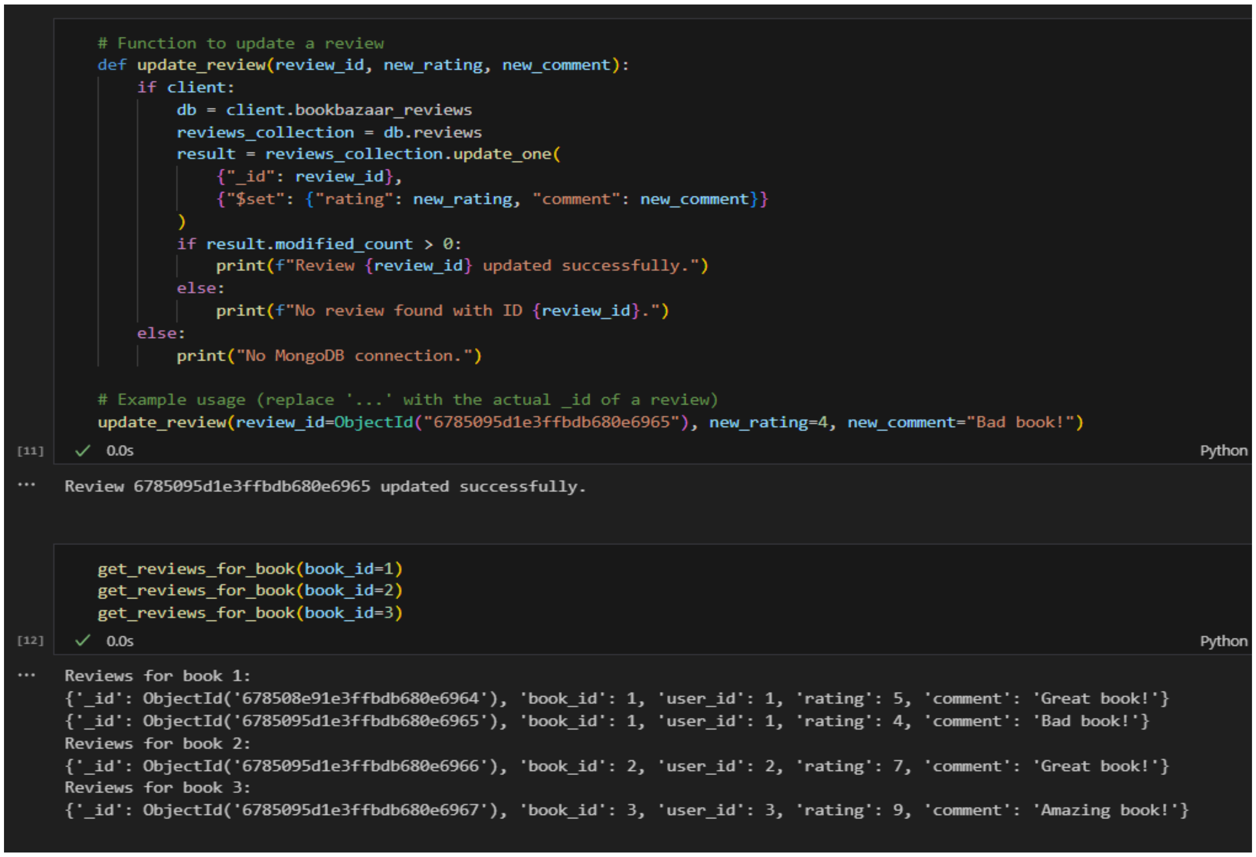


```
# Function to retrieve reviews for a specific book
def get_reviews_for_book(book_id):
    if client:
        db = client.bookbazaar_reviews
        reviews_collection = db.reviews
        reviews = list(reviews_collection.find({"book_id": book_id}))
        print(f"Reviews for book {book_id}:")
        for review in reviews:
            print(review)
    else:
        print("No MongoDB connection.")

# Example usage
get_reviews_for_book(book_id=1)
get_reviews_for_book(book_id=2)
get_reviews_for_book(book_id=3)
```

[10] ✓ 0.0s

```
Reviews for book 1:
{'_id': ObjectId('678508e91e3ffbdb680e6964'), 'book_id': 1, 'user_id': 1, 'rating': 5, 'comment': 'Great book!'}
{'_id': ObjectId('6785095d1e3ffbdb680e6965'), 'book_id': 1, 'user_id': 1, 'rating': 5, 'comment': 'Good book!'}
Reviews for book 2:
{'_id': ObjectId('6785095d1e3ffbdb680e6966'), 'book_id': 2, 'user_id': 2, 'rating': 7, 'comment': 'Great book!'}
Reviews for book 3:
{'_id': ObjectId('6785095d1e3ffbdb680e6967'), 'book_id': 3, 'user_id': 3, 'rating': 9, 'comment': 'Amazing book!'}
```



```
# Function to update a review
def update_review(review_id, new_rating, new_comment):
    if client:
        db = client.bookbazaar_reviews
        reviews_collection = db.reviews
        result = reviews_collection.update_one(
            {"_id": review_id},
            {"$set": {"rating": new_rating, "comment": new_comment}}
        )
        if result.modified_count > 0:
            print(f"Review {review_id} updated successfully.")
        else:
            print(f"No review found with ID {review_id}.")
    else:
        print("No MongoDB connection.")

# Example usage (replace '...' with the actual _id of a review)
update_review(review_id=ObjectId("6785095d1e3ffbdb680e6965"), new_rating=4, new_comment="Bad book!")
```

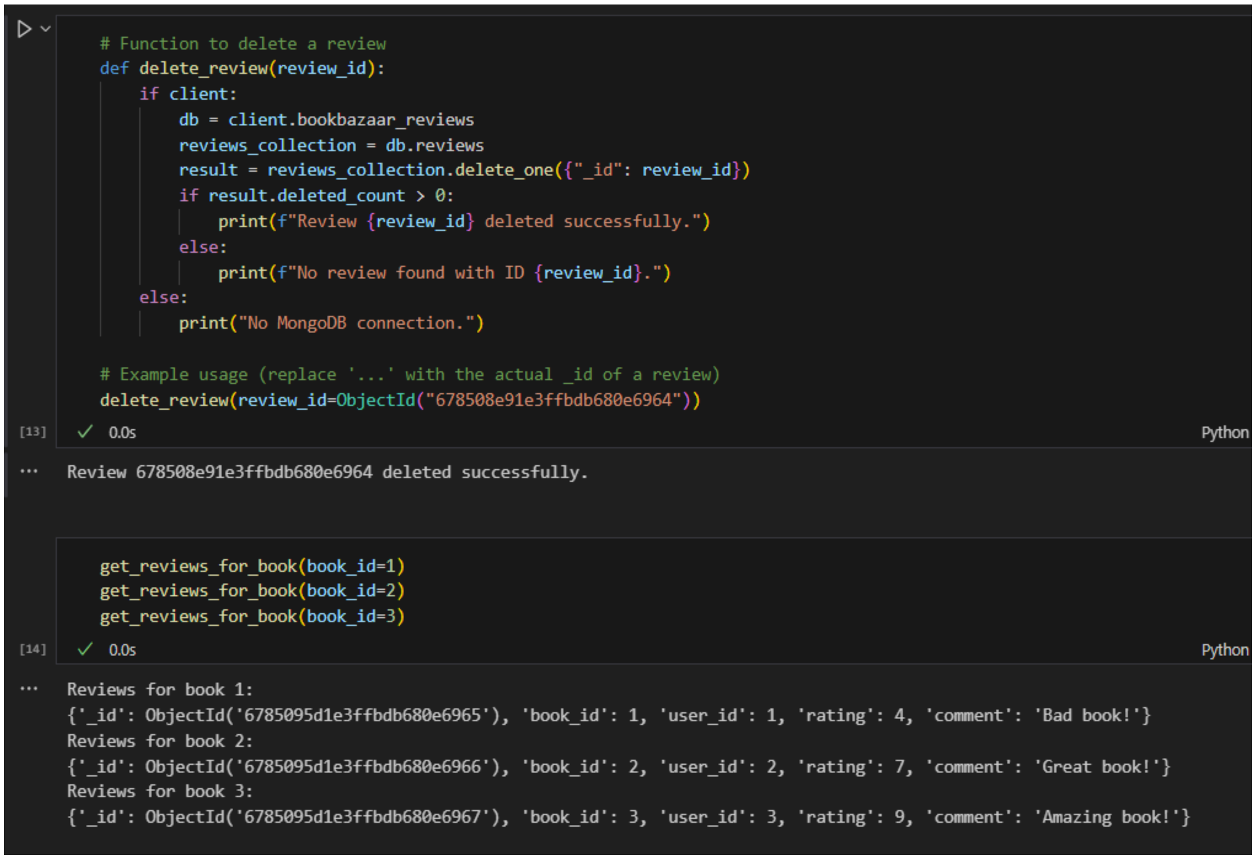
[11] ✓ 0.0s Python

```
Review 6785095d1e3ffbdb680e6965 updated successfully.
```

```
get_reviews_for_book(book_id=1)
get_reviews_for_book(book_id=2)
get_reviews_for_book(book_id=3)
```

[12] ✓ 0.0s Python

```
Reviews for book 1:
{'_id': ObjectId('678508e91e3ffbdb680e6964'), 'book_id': 1, 'user_id': 1, 'rating': 5, 'comment': 'Great book!'}
{'_id': ObjectId('6785095d1e3ffbdb680e6965'), 'book_id': 1, 'user_id': 1, 'rating': 4, 'comment': 'Bad book!'}
Reviews for book 2:
{'_id': ObjectId('6785095d1e3ffbdb680e6966'), 'book_id': 2, 'user_id': 2, 'rating': 7, 'comment': 'Great book!'}
Reviews for book 3:
{'_id': ObjectId('6785095d1e3ffbdb680e6967'), 'book_id': 3, 'user_id': 3, 'rating': 9, 'comment': 'Amazing book!'}
```



```
# Function to delete a review
def delete_review(review_id):
    if client:
        db = client.bookbazaar_reviews
        reviews_collection = db.reviews
        result = reviews_collection.delete_one({"_id": review_id})
        if result.deleted_count > 0:
            print(f"Review {review_id} deleted successfully.")
        else:
            print(f"No review found with ID {review_id}.")
    else:
        print("No MongoDB connection.")

# Example usage (replace '...' with the actual _id of a review)
delete_review(review_id=ObjectId("678508e91e3ffbdb680e6964"))
```

[13] ✓ 0.0s Python

```
Review 678508e91e3ffbdb680e6964 deleted successfully.
```

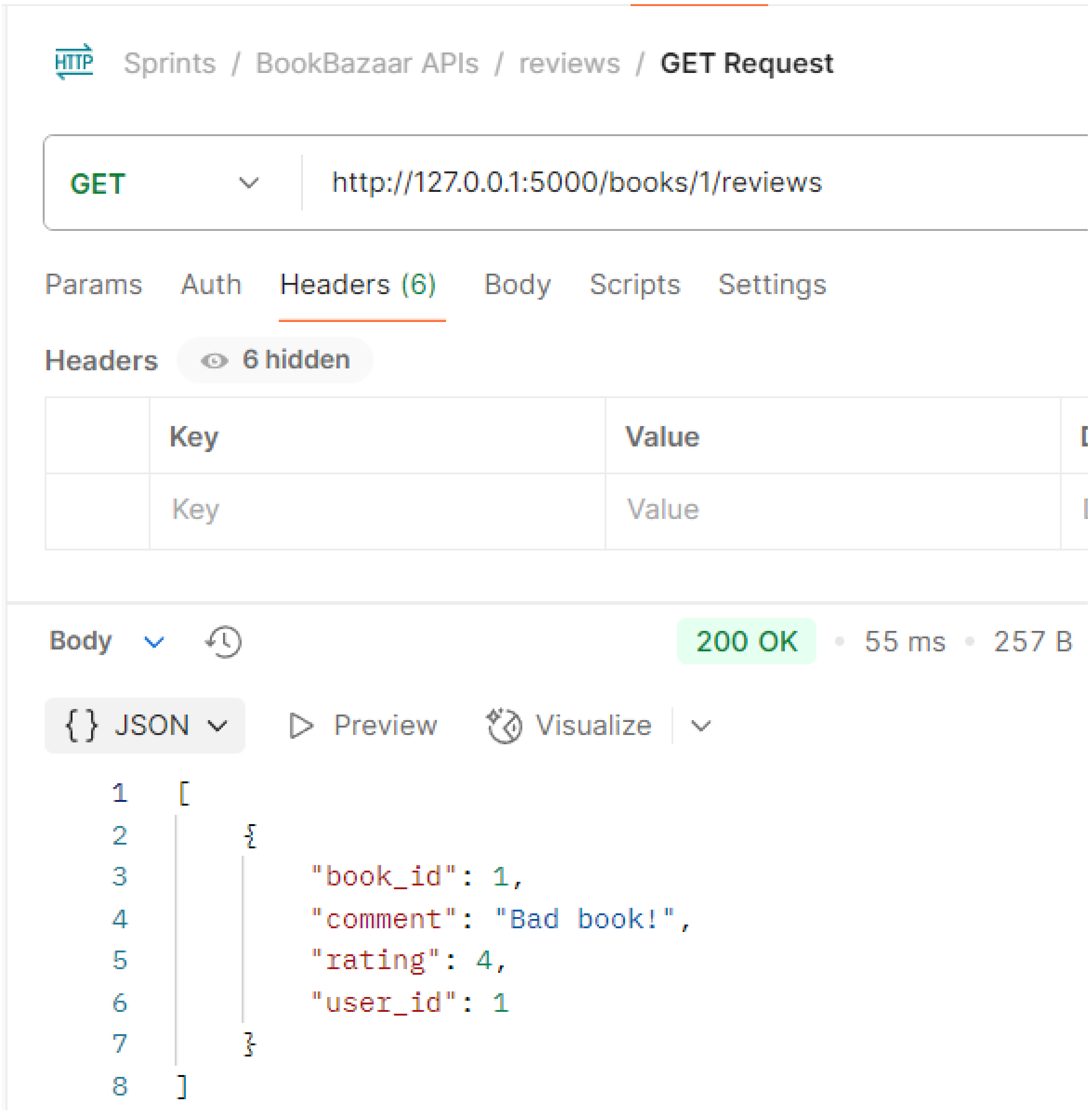
```
get_reviews_for_book(book_id=1)
get_reviews_for_book(book_id=2)
get_reviews_for_book(book_id=3)
```

[14] ✓ 0.0s Python

```
Reviews for book 1:
{'_id': ObjectId('6785095d1e3ffbdb680e6965'), 'book_id': 1, 'user_id': 1, 'rating': 4, 'comment': 'Bad book!'}
Reviews for book 2:
{'_id': ObjectId('6785095d1e3ffbdb680e6966'), 'book_id': 2, 'user_id': 2, 'rating': 7, 'comment': 'Great book!'}
Reviews for book 3:
{'_id': ObjectId('6785095d1e3ffbdb680e6967'), 'book_id': 3, 'user_id': 3, 'rating': 9, 'comment': 'Amazing book!'}
```

Task 11:

Integrate MongoDB Operations into the APIs



Sprints / BookBazaar APIs / reviews / GET Request

GET http://127.0.0.1:5000/books/1/reviews

Params Auth Headers (6) Body Scripts Settings

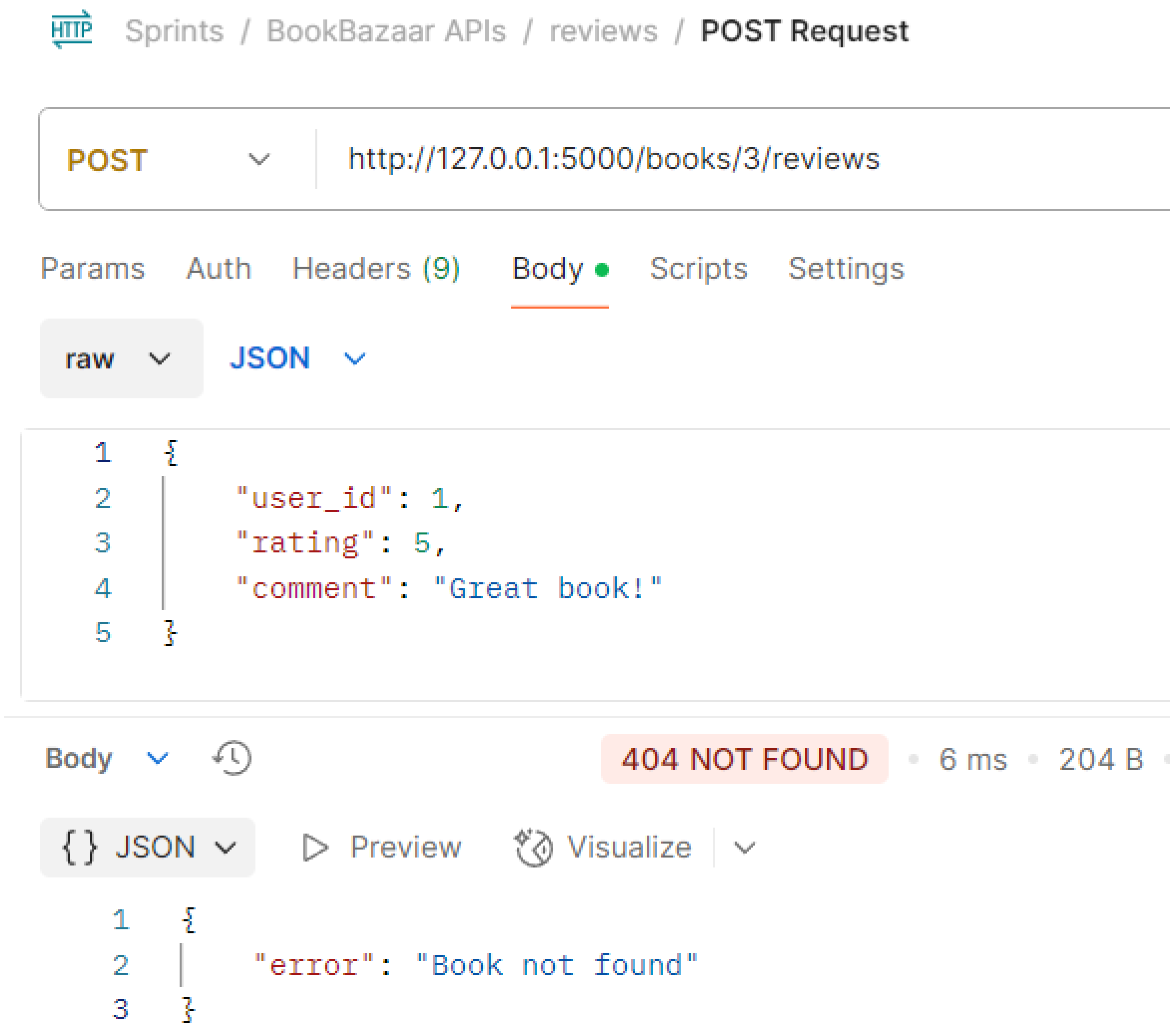
Headers 6 hidden

Key	Value
Key	Value

Body 200 OK • 55 ms • 257 B

{ } JSON Preview Visualize

```
[
    {
        "book_id": 1,
        "comment": "Bad book!",
        "rating": 4,
        "user_id": 1
    }
]
```



Sprints / BookBazaar APIs / reviews / POST Request

POST http://127.0.0.1:5000/books/3/reviews

Params Auth Headers (9) Body Scripts Settings

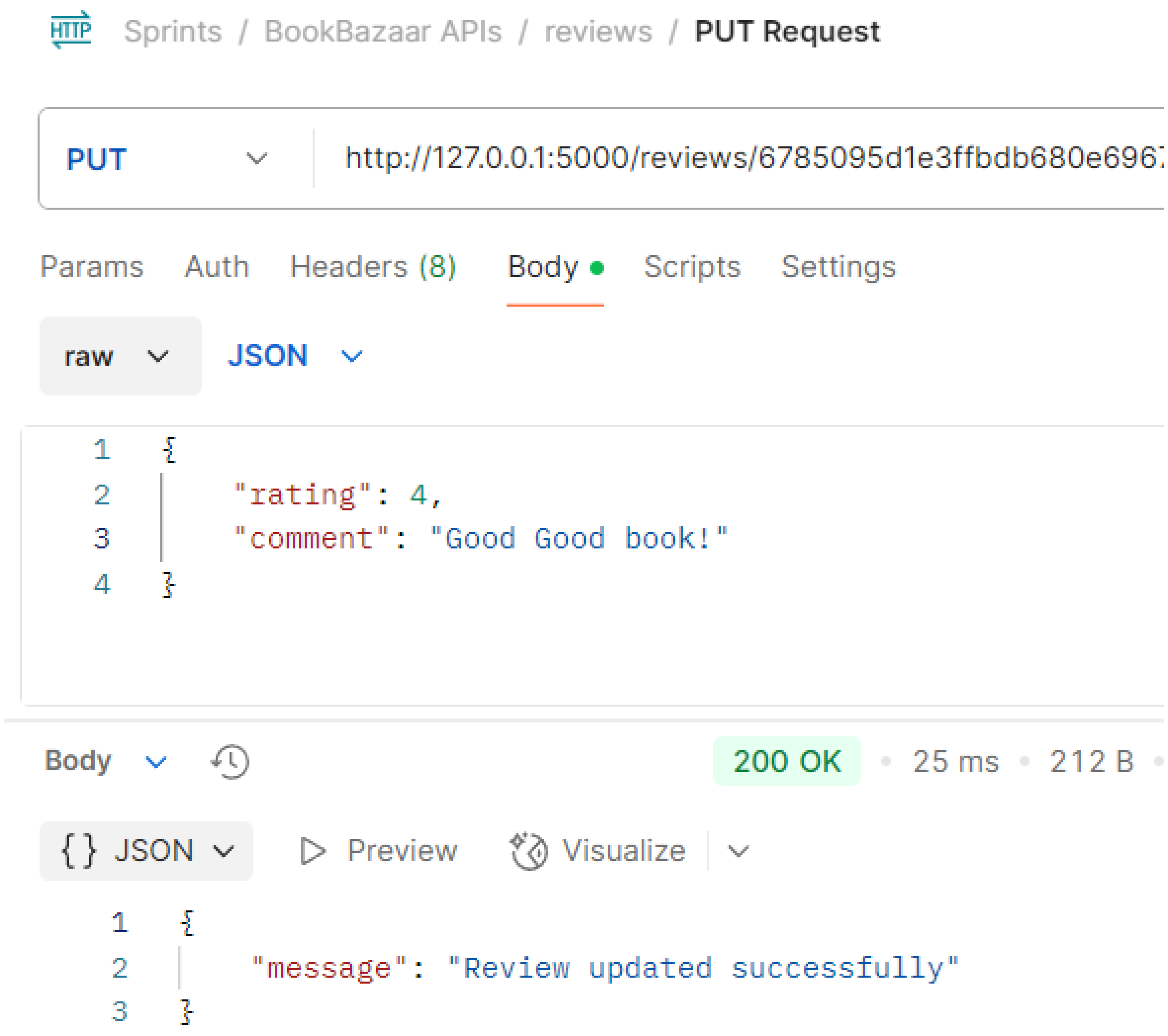
raw JSON

```json
{
    "user_id": 1,
    "rating": 5,
    "comment": "Great book!"
}
```

Body 404 NOT FOUND • 6 ms • 204 B

{ } JSON Preview Visualize

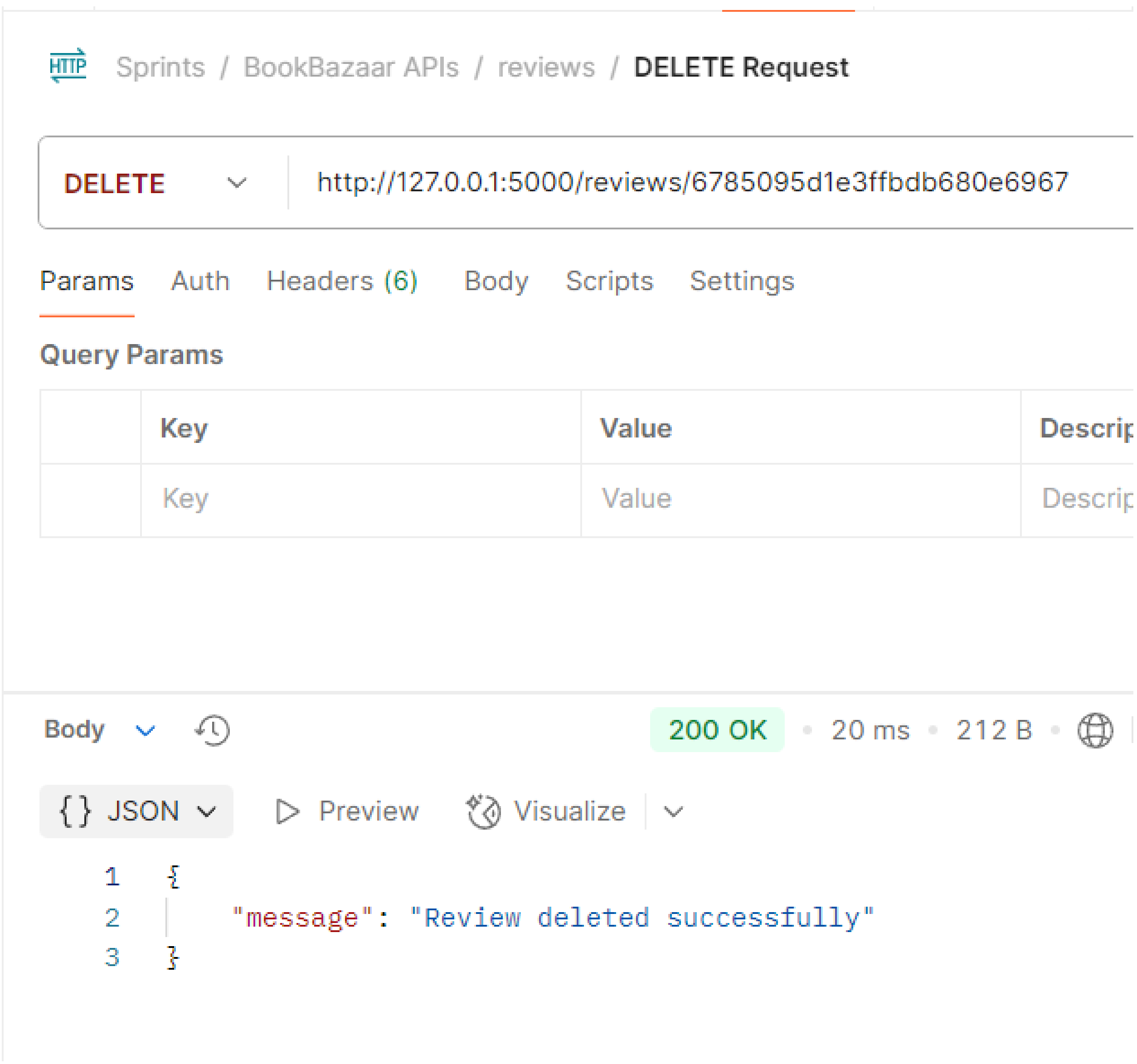
```json
{
    "error": "Book not found"
}
```



http://127.0.0.1:5000/reviews/6785095d1e3ffbdb680e696

```
{
    "rating": 4,
    "comment": "Good Good book!"
}
```

```
{
    "message": "Review updated successfully"
}
```


Sprints / BookBazaar APIs / reviews / DELETE Request

DELETE http://127.0.0.1:5000/reviews/6785095d1e3ffbdb680e6967

Query Params

	Key	Value	Descrip
	Key	Value	Descrip

Body 200 OK • 20 ms • 212 B

```
{
    "message": "Review deleted successfully"
}
```